Loughborough University

This item was submitted to Loughborough's Research Repository by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

A lightweight Web GUI specification and realisation system and its impact on accessibility

PLEASE CITE THE PUBLISHED VERSION

PUBLISHER

© IEEE

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Stone, R.G.. 2019. “A Lightweight Web GUI Specification and Realisation System and Its Impact on Accessibility”. figshare. https://hdl.handle.net/2134/2628.

Loughborough University

This item was submitted to Loughborough's Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/

A lightweight Web GUI specification and realisation system and its impact on accessibility

Roger Stone
Department of Computer Science
Loughborough University of Technology, LE11 3TU, England
R.G.Stone@lboro.ac.uk

Abstract

Developments like XFORMS are supposed to encourage the web programmer to concentrate on the specification of the functionality of the web GUI rather than its appearance on screen. Instead of having the document delivery system make the same realisation choices for every user it could be better to give the user some control in order to fully exploit this degree of choice. This would be particularly important for disabled users. This work shows how a functional specification of a GUI may be rendered in different ways to different users by using personal preferences residing in a user's profile. This extends previous work on profile-based web document delivery. Because the GUI parts of pages are rendered according to their own personal preferences, the web pages become more accessible to disabled users with very much reduced effort from the author of the pages. The technique does not require a specific or modified browser and can be easily implemented using a combination of common technologies.

1. Introduction

It is claimed that a large percentage of the time needed to create an application is spent on the user interface, the 'GUI' [1]. For web pages the table-based, pixel-positioning methods of the early days have largely given way to CSS-based methods. There is a growing acceptance of elastic interfaces [2] – where the designer accepts that the user may well not see the interface rendered as the designer saw it. This could be because of a drastic change of screen size (e.g. desk top to mobile phone) or a substitution of styles (e.g to provide better contrast because the user has a visual impairment). One bold approach to reduce the time spent by the author on the user interface was to try and evolve it by genetic programming [3]. The development of XFORMS [4] reinforces the idea that the designer should simply be able to specify the interface and leave the realisation up to the browser. This provides a degree of freedom for the rendering engine. An example of the kind of choice that needs to be made is how to render on screen an interface element that has been specified as a one-out-of-n choice. The expectation is that under certain conditions a group of radio buttons would be chosen and under other conditions a select/menu element would be chosen. The question that remains is just how the rendering decisions are to be made. The conventional solution is to try and make decisions which would suit all users thus perpetuating the historical one-realisation-suits-all concept. An alternative, explored here, is to use the freedom offered by the designer to allow the delivery system to personalise the rendering of a page for a user by taking account of their preferences expressed in a profile.

The discussion about utilising degrees of freedom to benefit individual users can be applied to increase the accessibility of web pages and thereby benefit disabled users. WCAG [5] contains many tests to make sure that HTML Forms are accessible to all users. The conventional process is to try to ensure that a single realisation of a web page is accessible by all users (the

one-realisation-suits-all concept). However the new draft of the Web Content Accessibility Guidelines (WCAG 2.0 [6]) identifies specific disabilities and identifies which tests are intended to make sure that pages are accessible to people with those disabilities. This acknowledges the differences in user requirements but persists in trying to capture their need by group or category. A more radical approach is to acknowledge the individual nature of each disabled user and try to render a page optimally for each individual. This idea has resulted in the creation of a profile for each user so that each page can be rendered taking into account the various preferences stored in the profile. So far the work on user profiling has captured three areas of preferences [7]. It is proposed to add the GUI preferences to the profile and thus fully exploit the degrees of freedom in the realisation of the interface.

The designers and implementers of XFORMS have been aware of accessibility issues and have taken some steps towards accessibility in the conventional sense. The designers are working to a charter to ensure that XForms meets W3C accessibility goals [8]. The accessibility of the XFORMS extension of the Firefox browser from Mozilla is to be improved [9]. However the approach to providing accessibility in both cases is via the one-realisation-suits-all concept.

In order to investigate the alternative concept a prototype system was constructed. The system was built to accept web pages containing interfaces captured in the specification style and to deliver such pages to an individual with interfaces dependent on the profile for that individual. It was convenient to use a basically HTML-based system but with the GUI specified in the notation of the User Interface component of XFORMS. An XSLT [10] stylesheet is used to render the XFORMS interface elements into HTML according to the user preferences as specified in a profile. Some of the implementations of XFORMS also use this transformation approach (e.g. [11]) and there are reverse transformations available (e.g. [12]) which try to assist with the legacy problem of converting HTML Forms to XFORMS.

2. User Profiles

Previous work on profile-based document delivery concerned profiles containing information about user preferences in relation to three aspects of a document: Styling, Essentiality, Accessibility checking.

In regard to styling, the profile stores text-size, text-font, colour preferences, etc. (c.f. TechDis Toolbar [13], Web Adaptation software [14]). In regard to essentiality, the profile stores an essentiality level (currently on a scale of 1-10) as a measure of how much information the user and/or their chosen browsing tool can comfortably handle in a single transaction. This aspect relies on the pages having been specially marked up by the author. We use a system of micro-formatting so that the conformity of the page to markup standards is not compromised. For example

```
<p class="ess10">This paragraph is crucially important.</p>
<p class="ess1">This is purely aesthetic content</p>
```

In regard to accessibility checking we consider that having an accessibility badge at the foot of a page is of very limited help to the disabled user. We maintain that a live check, made before the page is offered to the user, is of greater benefit. However in keeping with the concept of user-profiles it is realised that not all of the WCAG checks are of interest to all users. Therefore the accessibility checker only runs the tests requested in the profile. If any tests fail then a report is given to the user before they read the page so that they are fore-warned of any accessibility issues. We use our own accessibility checker written in CDuce inspired by Centeno [15].

It is into this environment that we wish to add the preferences of a user in relation to the rendering of the GUI of a web Form. These preferences could be expressed at different levels. They could merely be stylistic in relation to font-size for example. They could relate to a particular method of labelling of form controls (e.g. label preceding, use of square brackets, colon) as in this radio example [where, for convenience the radio buttons are represented by (o)]

```
[ male: (o) ] [ female: (o) ]
```

At these levels the preferences can be handled using CSS. However at the highest level, the preferences could be a specification of the circumstances in which to choose between a group of radio buttons and a select/menu element. It is at this level that it is proposed to use XSLT styling.

3. XFORMS

If coming to XFORMS from a background in HTML Forms a key difference is the concentration on functionality rather than rendering. So for example a

select1 tag can be used to specify an “exactly one from many” selection which may finally be rendered to the user as a radio group or as a menu selection, e.g.

```
<select1 ref="semester">
<label>Semester</label>
<item><value>1</value><label>Semester 1</label></item>
<item><value>2</value><label>Semester 2</label></item>
</select1>
```

Similarly a *select* tag can be used to specify a “none, one or many” selection which may finally be rendered to the user as a checkbox group or as a (multiple choice) menu selection, e.g.

```
<select ref="cards">
<label>cards held</label>
<item> <value>visa</value> <label>Visa</label> </item>
<item> <value>delta</value> <label>Delta</label> </item>
<item> <value>switch</value> <label>Switch</label> </item>
</select>
```

The *input* tag in XFORMS is only used when text input is required. A submit button is specified by the *submit* tag and a password by the *secret* tag.

4. XFORMS and Accessibility

XFORMS implementations via HTML have the opportunity to increase the accessibility of web forms in the sense that they can apply the WCAG rules consistently for all Form elements. Thus the WCAG guideline 12, Checkpoint 4, “Associate labels explicitly with their controls” can be enforced by making sure that the transformation from XFORMS to HTML picks up the label tag in the XFORMS specification and employs it in the HTML realisation. For example the first *item* in

```
<select1 ref="semester">
<label>Semester</label>
<item><value>1</value><label>Semester 1</label></item>
<item><value>2</value><label>Semester 2</label></item>
</select1>
```

could be processed (to produce a *label* tag with a attribute named *for*, relating to a radio button) and become

```
<label for="id001">Semester 1</label>
<input type="radio" name="Semester" id=" id001" value="1">
```

Furthermore these generated tags could be given class attributes (e.g. class="label" and class="labelled" so that they could be styled appropriately by a CSS stylesheet.

5. XFORMS and XSLT

It is quite easy to write an XSLT stylesheet to perform the basic processing outlined above to transform XFORMS Form elements into HTML Form elements. Perhaps one of the harder issues is to capture the pre-selection in the style of XFORMS. The information about pre-selected items is given in *instance* tags in the head of the document. For example, for the semester example above a pre-selection of semester two could be achieved by writing the *instance* tag

```
<instance>
  <data xmlns=""><semester>2</semester></data>
</instance>
```

When building select tags, radio buttons and check boxes this information has to be accessed and turned into *selected* or *checked* attributes.

The degree of freedom of presenting a one-out-of-many choice as a radio group or a select can be handled by a template match along the lines shown below

```
<xsl:apply-templates match="select1">
  <xsl:choose>
   <xsl:when test="count(item)<'6' ">
    ...transform XFORMS select1 to an HTML radio group...
   </xsl:when>
   <xsl:when test="count(item)>='6' ">
    ... transform XFORMS select1 to an HTML select...
   </xsl:when>
  </xsl:choose>
</xsl:apply-templates>
```

Here if there are less than six choices to be presented then a radio template is used, otherwise a select (menu) template is used.

6. XFORMS and Actions

It is proposed to translate an XFORMS specification of a GUI to XHTML for presentation to the user. During the translation the user profile will be consulted as to which realisation to use for each element. However a potential problem arises here. If the author of the GUI does not know which interface elements will be used, how can they write the script which is used to process the form results after submission?

For the one scripting system that has been explored in detail (PHP [16]) there is a simple solution. For the interface elements textbox, password, textarea, radio

button, select (single) and submit button the ‘value’ is passed as a single variable. All that has to be ensured is that the XSLT transformer translates the *ref* attribute of the XFORMS to the *name* attribute of the HTML version. The action script will then receive an instantiated variable of the same name and can then process the value in the normal way.

More care has to be exercised over the interface elements checkbox and select (multiple). Here the ‘value’ is a multiple value conveniently carried by an array in the processing script. Now PHP has a built-in behaviour of expanding an array to accept another value when empty indexing brackets are provided so that if the first reference to an array *$v* in a script is

```
$v[]="A"; $v[]="B"; $v[]="C";
```

then it has the same effect as

```
$v[0]="A"; $v[1]="B"; $v[2]="C";
```

This is exploited in the XSLT transformation of the XFORMS *select* element where the transformation of the *ref* attribute *r* is to a *name* attribute *r[]*. Thus an XFORMS *select* tag such as

```
<select ref="sport">
   <item><value>f</value><label>football</label></item>
   <item><value>r</value><label>tennis</label></item>
   ....
</select>
```

is transformed according to user preference to either the HTML menu

```
<select name="sport[]"  multiple="multiple" id="select">
   <option value="f">football</option>
   <option value="t">tennis</option>
   ...
</select>
```

or the HTML radio group

```
<label for="id1">football</label>
<input type="checkbox" name="sport[]" value="f" id="id1" />
<label for="id2">tennis</label>
<input type="checkbox" name="sport[]" value="t" id="id2" />
   ...
```

In either case the script programmer can use the built-in iterator each(...) to look through the array *$sport* to find out if it contains any of the values “f” or “t”, etc. as defined by the specification.

7. CSS and Profiles

Within the profile work it has already been established how to store CSS type styles in a user profile and apply them routinely on behalf of a user. To some extent this can be done with XFORMS. For example the styles *label* and *labelled* referred to earlier could be defined as

```
.label:before { content: " [ " }
.label:after { content: " --> " }
.labelled:after { content: " ] " }
```

This would have the effect of `bracketing' the label to the control and `pointing' from the label to the control. The effect on a radio button group would be something like

```
[ Semester 1--> (o) ] [ Semester 2--> (o) ]
```

This would help overcome the potential ambiguity in a long list of radio buttons as to whether the label was on the left or the right.

```
mon (o) tues (o) wed (o) thurs (o) fri (o) sat (o) sun (o)
```

However a much more drastic `styling' can be contemplated. If a particular disabled user can always operate radio buttons confidently (even if badly styled) and has difficulty with menus (however they are styled), then that user should be able to state as a preference that all one-out-of-many choices are to be presented using radio buttons. This level of styling cannot be achieved with CSS and so it is necessary to resort a more powerful styling system.

8. XSLT and Preferences

The Extensible Style Language XSL contains XSLT which is a transformation language. A stylesheet is constructed out of templates which are applied when they match some criterion. Typically this criterion is based solely on the XML input that is being processed (e.g. “is there a descendant of the root node of the input which has a certain tag name and attribute value...?”). For the situation under consideration the criteria will be based on the web page requested by the user but will also include preferences expressed by the user in their profile.

Traditionally radio buttons and checkboxes are used for small numbers of choices and menus are used for the larger numbers of choices. This means that the basic choice that a user might make is to decide on the

break point at which changeover happens. Thus a template for *select1* might be coded to produce alternative results which depend on the value of an *<xsl:variable> USER_RADIO_MAX* as follows

```
<xsl:template match="select1">
   <xsl:if test="count(item)<$USER_RADIO_MAX">
      <xsl:apply-templates mode="radio" />
   </xsl:if>
   <xsl:if test="count(item)>=$USER_RADIO_MAX">
      <xsl:apply-templates mode="menu" />
   </xsl:if>
</xsl:template>
```

so that an XFORMS *select1* element is either transformed into a radio group (if there are less than six items) or a menu if there are more items.

9. Storing GUI preferences in a profile

Now that XSLT gives the ability to transform the web page according to user preferences it must be established how to provide the correct stylesheet which can transform the page according to user preferences. First the range of preferences that need to be accommodated should be established.

The phrase “radio to 6, checkbox to 10” could be treated as a specification of user preferences which would suggest switching to menus for a *select1* tag with more than 6 and for a *select* tag with more than 10 options. The specification “radio to 0, checkbox to 0” would be treated as a request to always use menus. The specification “radio to 65535, checkbox to 65535” would effectively be treated as a request never to use menus.

The appearance of the large number (65535) may bring to mind a question as to how many choices there can legitimately be in a selection. One of the common occasions when a relatively large number of choices is met is the traditional sign-up page where a user is giving personal details in order to register for some kind of service. This will often have a form element which contains an alphabetical choice of country and typically this will have around 250 entries. This can be an infuriating choice to make, even as an able-bodied user, if you are not sure whether your country is listed under “b” for Britain, “e” for England or “u” for United Kingdom. Neither radio buttons nor a menu seem ideal for this wide choice. It is possible that a further interface option might be invented to try to the effort required to make this choice - maybe a text box with a paragraph of hints following to suggest what to type in the text box:

```
Country: [type here] one of AL(Albania)
   ... AR(Argentina) ... AU(Australia), AT(Austria)
   ... EN(England) ... FR(France) ...
```

In fact there are more possibilities even within radio buttons, checkboxes and menus. XFORMS uses the *appearance* attribute with three possible values “full”, “compact” or “minimal”. This allows the designer to hint at a suitable realisation. Thus a 4-way *select1* could be portrayed as a radio if “full” is specified, as a stay-open 4 choice menu if “compact” is specified and as a collapsing menu of height 1 for “minimal”. However these are choices that we would want the user to make.

In the preceding section there was a discussion of using CSS styling to introduce content onto the page to help ‘bracket’ together a form element and its label. However CSS Styling cannot switch the order of a label and the element it labels. Having label first or element first is another preference that must be handled within the XSLT styling.

Because of the range and detail of the interface preferences it was decided to experiment first with associating complete stylesheets with individual users rather than trying to parameterise a master stylesheet. It is not envisaged that a user would have to write a stylesheet or any part of it directly. Rather they should be able to interact with a visual tool which offers a range of basic settings which they could customise.

10. Implementation details

The environment for this work is largely (X)HTML 1.0 with dynamic pages implemented in PHP with mySQL providing database support. This means that introducing XFORMS requires the rewriting of HTML Forms into equivalent XFORMS specifications. In this lightweight, experimental implementation using only the user interface part of XFORMS, the XFORMS tags are written directly into an otherwise XHTML 1.0 compliant file. An XSLT stylesheet recognises and rewrites the XFORMS parts and passes the remaining XHTML through unchanged. The details are slightly different for static or dynamic pages. Where the original HTML Form-based version was a static page the XFORMS version can be transformed by a very short server-side script which simply identifies the xml file and the xsl transformation file and invokes the xslt processor.

Input	Processor	Output

XFORM.xml	PHP Script invoking xslt processor (+ templates.xsl)	HTML Form

Table I - Static Page

Where the original HTML Form-based version was a dynamic page, a new dynamic page is created which generates XFORMS output in place of HTML Form output. A new short server-side script is then written which first executes the XFORMS-updated script using wget and then transforms the result using the xslt processor.

Input	Processsor 1	Processor 2	Output
XFORM.php	wget	xslt processor (+ templates.xsl)	HTML

Table II - Dynamic Page

The profiles are stored in a mySQL database table. The profile-based delivery tool we have extended is referred to as a filter. By starting at the filter's home page and after selecting a profile to use, the user chooses their first URL to browse. The filter makes changes to the page in accordance with the preferences of the user and hopefully causes an optimum page to be rendered to the user. If the user now chooses to select a link away from the page then the new page will also be filtered. This is because part of the action of the filter is to rewrite links. An original link that might have looked like

```
<a href="link_URL">link text</a>
```

is rewritten to be re-routed through the filter as

```
<a href="filter?url=link_URL&pref=pref_URL">link text</a>
```

11. Conclusions

There are two opposing views about ensuring accessibility of web pages. In the one view the page author is expected to create a single version of their page which is accessible by all. In the other view there is a tacet admission that this one-realisation-suits-all concept is flawed. The alternative is to provide a system which knows enough about the person for whom it is providing pages so that it can adapt the page in an optimal way for the user. The work reported here is pursuing this second alternative by storing knowledge about the user in a profile. Various kinds of preference have been considered for inclusion in the profile and the most recent is the preferences associated with the web GUI elements. In order to give the filter opportunities to offer the user different versions of the GUI, the original page must be specified at a higher level than is normal in HTML. For this study the notation of XFORMS was used as it concentrates on the functionality required of the interface rather than the visual realisation. An XSLT transformation is used to translate the XFORMS specification of the GUI into XHTML according to the preferences stored in the user profile. Thus the technique presented does not require a specific or modified browser and can be easily implemented using a combination of common technologies.

The work required now is to conduct user trials to see if users report the expected benefits and to decide on the best way to store the GUI preferences for a user. In a more general sense there is the general search for any other way in which the profile-based system can improve document delivery on the web.

References

[1] B.A. Meyers and M.B. Rosson, “Survey on User Interface Programming”, Proceedings of the Conference on Human Factors in Computing Systems, 1992, http://portal.acm.org/citation.cfm?id=142789

[2] P. Griffiths, “Elastic Design,” http://www.alistapart.com/articles/elastic/

[3] M.S. Withall, “The Evolution of Complete Software Systems,” PhD Thesis, http://www-staff.lboro.ac.uk/ mpmsw/

[4] W3C-XFORMS, “XFORMS 1.0 (Second Edition) - W3C Recommendation”, 2006, http://www.w3.org/MarkUp/Forms/

[5] W. Chisholm and G. Vanderheiden, “Web Content Accessibility Guidelines 1.0,” Web Accessibility Initiative (WAI), World Wide Web Consortium, 1999, http://www.w3.org/TR/WAI-WEBCONTENT/

[6] B. Caldwell, W. Chisholm, J. Slatin and G. Vanderheiden, “WCAG 2.0, Web Content Accessibility Guidelines 2.0 (Working Draft),” Web Accessibility Initiative (WAI), World Wide Web Consortium, 2005, http://www.w3.org/TR/WCAG20/

[7] J. Dhiensa, C.H.C. Machin, F. Smith and R.G. Stone, “Optimizing the User Environment: Leading Towards an Accessible and Usable Experience,” Accessible Design in the Digital World Conference 2005, Dundee, Scotland. 23 - 25 August 2005, http://ewic.bcs.org/conferences/2005/accessible/session1/paper3.htm

[8] XFORMS Working Group Charter,

http://www.w3.org/MarkUp/Forms/2003/xforms-wg-charter.html

[9] XFORMS and Firefox: Accessibility Grant, http://www.beaufour.dk/blogarchives/

2006/05/xforms accessib.html

[10] W3C-XSLT, “XSL Transformations (XSLT) 1.0 - W3C Recommendation,” 1999, http://www.w3.org/TR/xslt

[11] Orbean Presentation Server, http://www.orbeon.com/ops/doc/integration-xforms-jsp

[12] XHTML to XForms converter, http://sourceforge.net/projects/xhtml-toxforms

[13] P. Rainger, “User Preferences Toolbar,” TechDis, http://www.techdis.ac.uk/index.php?p=1 20051905100544

[14] V.L. Hanson, “The user experience: designs and adaptations,” Proceedings of the 2004 international cross-disciplinary workshop on Web accessibility (W4A), New York City, USA, http://portal.acm.org/citation.cfm?id=990659

[15] V. L. Centeno, et al., “Web Accessibility Evaluation Tools: A Survey and Some Improvements”, Electronic Notes in Theoretical Computer Science, article 17, 157(2), Springer ENTCS, 1st International Workshop on Automated Specification and Validation of Websites (WWV 2005), Valencia, Spain, 2005, pp. 87-100

[16] R. Lerdorf, “Hypertext Preprocessor (PHP),” http://www.php.net/